Cross-16
Meta-Assembler User's Guide

version 2.0

UNIVERSAL CROSS-ASSEMBLERS

P.O. Box 6158, Saint John, N.B. Canada E2L 4R6 Voice/Fax (506) 847-0681

"Unauthorized use, duplication or distribution is strictly prohibited"

Cross-16
Meta-Assembler User's Guide

version 2.0

Universal Cross-Assemblers
P.O. Box 6158
Saint John, N.B.
Canada E2L 4R6
Voice/Fax: (902)847-0681

Copyright (C) 1989 by Universal Cross-Assemblers.
All rights reserved.

License

Use of this product by any person other than the purchaser or on more than one keyboard of any computer at any specific time is strictly prohibited.

Warranty

Universal Cross-Assemblers will replace the assembler disk and/or reference manual once for a period of ninety days from the date of purchase if the original disk and/or user's guide are unreadable upon their return. The purchaser is urged to make a back-up copy of the supplied disk. If the product does not meet the specifications published by Universal Cross-Assemblers, return it within thirty (30) days of our invoice date, with a written explanation of the problem, for a full refund.

Disclaimer

Universal Cross-Assemblers, disclaims all liabilities to any damages or losses that may occur through the use of its products or the breach of its warranty, other than the single replacement of the original product.

Acknowledgment

This program was compiled using Turbo C Version 2.0 which is the property of:

Borland International
1800 Green Hills Road
Scotts Valley, CA, USA
95066

This manual was written using Wordstar 2000 Plus Release 3, which is the property of:

Wordstar International Corporation
33 San Pablo Avenue
San Rafael, CA, USA
94903

Table of Contents

1. Introduction 5

2. Universal Truths Concerning Cross-16 6

3. Running Cross-16 7

4. The Cross-16 Assembly Source File 9
    4.1. Assembly Line Format 9
    4.2. Labels 10
    4.3. Operands 11
        4.3.1. Integer Constants 11
        4.3.2. String Constants 13
        4.3.3. Arithmetic Expressions 14

5. Cross-16 Assembler Directives 16
    5.1. CPU - Central Processing Unit Table Declaration 17
    5.2. DFB - Define Byte 18
    5.3. DFL - Define Long Integer 19
    5.4. DFS - Define Storage 20
    5.5. DWL - Define Word (Least Significant Byte First) 21
    5.6. DWM - Define Word (Most Significant Byte First) 22
    5.7. END - End of Source Program 23
    5.8. EQU - Equate Label 24
    5.9. HEX - Hexadecimal File Control 25
    5.10. HOF - Hexadecimal Output Format 26
    5.11. INCL - Include Source File 27
    5.12. LIST - List File Control 28
    5.13. ORG - Program Counter Origin 29
    5.14. PAGE - Page Control 30
    5.15. TITL - Title of Listing 31
    5.16. WDLN - Word Length 32

6. The Cross-16 Listing File 33
    6.1. Listing Format 33
        6.1.1. Assembly Error Codes 33

7. The Cross-16 Hexadecimal File 37

8. The Cross-16 Instruction Table 39
    8.1. Register Definition 40
    8.2. Operand Definition 41
    8.3. Addressing Mode Definition 43
    8.4. Mnemonic Definition 45
    8.5. Additional Table Features 49
    8.6. Checking the Instruction Table 49

9. Non-Assembly Error Messages 50
9.1. Warning Messages 51
9.2. Fatal Errors 53

10. Bugs 55

11. Processor Manufacturers 56

12. Registration 57

Index 58

1. Introduction

Congratulations! You have selected one of the most cost effective tools for developing assembly language programs for 4, 8, 16 and 32 bit microprocessors and microcontrollers. As supplied, Cross-16 will compile assembly language programs for over twenty different processors, and it may be expanded by the user to handle many more! But first a few words on assembly language programming in general.

An assembly language program is a text source file where the manufacturer's assembly instruction mnemonics or "memory aids" are used to directly represent the binary machine codes or opcodes that a computer actually executes. An assembler is a computer program that converts these mnemonics into a their corresponding binary codes. The simple word "assembler" usually refers to a resident or self-assembler, which assembles programs for the same processor it runs on. The Microsoft Macro Assembler for the 80x86 family running MS-DOS is an example of this. A cross-assembler, assembles programs for a single target processor which differs from the host processor. Examples of these, supplied by other software houses, may be found in the more technical computer and electronic periodicals.

Cross-16 from Universal Cross-Assemblers is a meta-assembler, in that it assembles programs for numerous different target processors. It reads an assembly language source file and a corresponding assembler instruction table from disk, and writes an assembled listing and a hexadecimal or binary machine code output file. By using a flexible instruction table structure, Cross-16 is designed to compile assembly language source code for most microprocessors and microcontrollers with an address word of 24 or less bits. To further enhance it's flexibility, Cross-16 will produce a machine code output file in binary and the Intel and Motorola 8 and 16 bit hexadecimal formats.

As one proceeds through this manual, one will pass from the broad generalities needed to use Cross-16, to the detailed specifics necessary to write a processor instruction table. If the user simply wishes to assemble code for one of the processors with an instruction table provided, then the user need not read chapter 8 on creating an instruction table, but should carefully study any notes provided in the example given for their target processor. To successfully tailor a provided table or create a new one, the user must study the entire manual, and perhaps several of the provided tables.

2. Universal Truths Concerning Cross-16

The following are broad generalities for those already familiar with compilers and assemblers. All of the following points will be explained in greater detail in the appropriate sections of this manual.

Cross-16 is a two pass cross-assembler with an optional third pass if a phase error is detected.

All input files, output files and processor tables contain ASCII characters with each line terminated by an ASCII carriage return and line feed.

Cross-16 expects the most significant bit of all ASCII characters to be clear (i.e. ASCII_CHAR = 0...127).

Only one processor assembly instruction or Cross-16 assembler directive is permitted per line.

All label declarations must be terminated with a colon ":".

All labels, expressions and operands are internally stored and manipulated using 32-bit signed integers.

Expression operators are similar in both format and precedence to the C programming language.

Cross-16 makes no distinction between upper and lower case alphabetic characters.

Blank lines in the assembly source code are reproduced in the assembly listing but otherwise ignored.

Overflow errors, undefined labels, and invalid expressions are assigned a value of zero.

There are no restrictions on the character length of labels or processor instructions and all characters are significant. However, Cross-16 input lines must not exceed 255 characters in length.

Cross-16 does not support or include a linker or librarian.

3. Running Cross-16

The enclosed disk contains the Cross-16 Meta-Assembler (C16.EXE), and over sixteen processor tables (cpu.TBL) with corresponding example assembly source files (Ecpu.ASM). Universal Cross-Assemblers recommends that the user backs up the entire Cross-16 disk. Then copy C16.EXE and the table and example file for your target processor to a disk or directory with your favorite program editor. The following examples use the E8048.ASM and 8048.TBL files.

Cross-16 may be run using the following command:

```
C16 sourcefile [-L listfile] [-H hexfile]
```

where the square brackets [] indicate optional items.

The "-L" instructs Cross-16 to produce an assembly list file using the immediately following file name. The "-H" tells Cross-16 to produce a machine code output file using the immediately following file name. If these are omitted the corresponding files will not be produced. All error codes will be displayed on the screen, regardless of whether a list file is created. The format of the hex file is set using the HOF directive, explained later in this manual. The processor instruction table used by Cross-16 is also specified in the source file using the CPU directive. The order in which the source, listing and hex files are specified in the command line does not matter. Disk and directory names may be included in the file names.

Some examples are:

```
C16 E8048.ASM
                              source:   E8048.ASM
                              listing:  NONE
                              hex:      NONE


C16 E8048.ASM -L B:E8048.LST

                              source:   E8048.ASM
                              listing:  B:E8048.LST
                              hex:      NONE


C16 C:E8048.ASM -H \HEX\E8048.HEX

                              source:   C:E8048.ASM
                              listing:  NONE
                              hex:      \HEX\E8048.HEX
```

```
C16 E8048.ASM -H E8048.HEX -L E8048.PRN

                              source:   E8048.ASM
                              listing:  E8048.PRN
                              hex:      E8048.HEX
```

Running Cross-16 produces the following on the screen:

```
C>C16 E8048.ASM


Cross-16 Meta-Assembler    PC/MS-DOS    Version 2.01
Copyright (C) 1989 Universal Cross-Assemblers

Starting Pass Number 1

Starting Pass Number 2

Checksum = 9668
End of Assembly -- No Errors

C>
```

where "C>" is the operating system prompt. Since Cross-16 is a two pass assembler, the source file is read twice. The processor instruction table is only read once and kept in RAM for the second pass. Assembly errors will be displayed on the screen during the second pass, in addition to being placed in the listing if one was specified. The listing and machine code files are written during the second pass only. The checksum is a byte by byte addition of the compiled binary code.

Execution of Cross-16 may be aborted any time by simultaneously pressing the Ctrl and "C" or the Ctrl and Break keys.

4. The Cross-16 Assembly Source File

The source file is the ASCII assembler source code to be assembled by Cross-16. From the source file, the processor instruction table is selected using the CPU directive, and the format of the hexadecimal file is set using the HOF directive.

The following sub-sections describe the fundamental building blocks which make up the assembly source file. Examples are given for each section, but these have been assembled and taken from the listing file. Therefore, the user does not provide the first 16 characters of these examples.

4.1. Assembly Line Format

Only one assembly instruction or assembler directive is permitted per line. The assembly line is free format, labels need not start in column 1. Each line may contain some or all of the following sequence of identifiers:

```
line#     label:    operation     operand(s)     ;comment
```

Where...

Line # is an optional decimal integer in the range of 0 to 65535 created by some editors representing the source code line number. If present, Cross-16 will ignore this field except to reproduce it in the listing.

Label is a phrase starting with an alphabetic ASCII character "A-Z" or an "_", "." or "?" and ending with a colon ":", which is assigned the present value of the program counter or other user specified value. Characters within the label must be alphanumeric "A-Z", "0-9" or an "_", "." or "?".

Operation is a Cross-16 assembler directive, or processor assembly instruction defined in the instruction table. All operations must start with an alphabetic "A...Z" character.

Operands are registers, labels, constants or expressions representing integer values in the range of -2,147,483,648 to 2,147,483,647 and/or character strings. They may be embodied within an assembly language instruction.

Comment is a statement following a semicolon ";" usually used to describe the assembly language program. The ";" may be placed anywhere on the assembly line and Cross-16 ignores all characters following it on that line.

4.2. Labels

A label is an alphanumeric series of characters representing an integer in the range -2,147,483,648 to 2,147,483,647. The label field must start with an alphabetic ASCII character "A-Z" or an "_", "." or "?" and ending with a colon ":", even when used with the EQU assembler directive. Characters within the label must be alphanumeric "A-Z, 0-9" or an "_", "." or "?". Except for the EQU directive, a label is optional, and assigned the current value of the program counter. Labels may be of any character length, except that a Cross-16 input line cannot exceed 255 characters, and all characters are significant. Cross-16 makes no distinction between upper and lower case characters. A label may also stand alone on a line, in which case it will be assigned the current value of the program counter. A dollar sign "$" may be used as an operand representing the current value of the program counter.

The following examples are taken from an actual Cross-16 listing. They use the equate (EQU) directive described later in the manual. Very simply, the equate (EQU) directive assigns the label on its left, the value of the expression on its right.

```
0200                    ORG  0200H     ; Origin

                ;Valid examples of labels are:
                ;
1234 =          STRT1:  EQU  1234H     ;Label on directive
1234 =          LD_UP:  EQU  1234H     ;Label with "_"
0200            alone:                 ;Stand alone label
0200            DITO:   EQU  $
0987 =          LONG_LONG_LONG: EQU 987H ;Any length
```

4.3. Operands

The following sub-sections describe numeric constants, string constants, and arithmetic operators which in combination with labels may be used to form operands.

4.3.1. Integer Constants

A integer constant is a series of ASCII digits representing a 32 bit signed integer in one of several number bases. Cross-16 supports the following three numeric constant formats:

1) C programming language (i.e. 0377 = 255 = 0xff)

If the first digit is a zero, the integer is taken to be Octal, and the remaining characters must be 0-7. If the first digit is a zero, immediately followed by an "x", the integer is taken to be hexadecimal, and the following digits must be 0-9 or a-f. Otherwise, the constant is taken to be decimal, and all the digits must be in the range of 0-9. The valid bases with their corresponding C language identifiers and character ranges are as follows:

Leading	Base		Characters
0	Octal	Base 8	0-7
1-9	Decimal	Base 10	0-9
0x	Hexadecimal	Base 16	0-9, A-F

2) Trailing alphabetic (i.e. 011111111B = 377Q = 255D = 0FFH)

The base of integer constant is indicated by a trailing alphabetic character. All constants must start with a numeric digit 0-9. The default base is base 10. Both "O" and "Q" may be used to specify octal numbers to avoid confusion between "0" and "O". The valid bases with their corresponding trailing alphabetic characters and character ranges are as follows:

Trailing	Base		Characters
B	Binary	Base 2	0-1
O	Octal	Base 8	0-7
Q	Octal	Base 8	0-7
D	Decimal	Base 10	0-9
H	Hexadecimal	Base 16	0-9, A-F

3) Leading dollar sign (i.e. 255 = $FF)

If the first digit is a dollar sign "$", the integer is taken to be hexadecimal, and the following digits must be 0-9 or A-F. The dollar sign by itself represents the current value of the program counter.

Some examples of numeric constants are:

```
                ;Valid examples of numeric constants are:
                ;
00AF =          BIN1:   EQU  10101111B  ;Binary
00FF =          OCT1:   EQU  377O       ;Octal
003C =          OCT2:   EQU  74Q        ;Octal
00FF =          OCT3:   EQU  0377       ;Octal C
FFFF =          DEC1:   EQU  -1         ;Decimal
000A =          DEC2:   EQU  10D        ;Decimal
00FF =          HEX1:   EQU  0FFH       ;Hexadecimal
00FF =          HEX2:   EQU  0xff       ;Hexadecimal C
00FF =          HEX3:   EQU  $FF        ;Hexadecimal
```

4.3.2. String Constants

String constants "string" consist of a series of ASCII characters between two quotation marks ("). Cross-16 will convert these constants to a hexadecimal representation of their ASCII values in the listing. Lower case characters within string constants will be represented as such. A quotation mark (") cannot appear within a character string, for it will be interpreted as being the end of that string. If a quotation mark is needed, Universal Cross-Assemblers recommends that an apostrophe be used ('). The user may also terminate the string, insert the binary value of a quotation mark (22H), and then start another string. A string constant may also be used as an operand where applicable. In a DFB statement, a string constant may be of any length, bearing in mind that an assembly source line must not exceed 255 characters in length. When used as an operand, or in the DWM, DWL, DFL or EQU directives, an error will be flagged if the string constant exceeds the length of the operand specified by the assembler directive. A string constant cannot be extended beyond it's present line without terminating the string with a quotation mark. Although only the first five or seven bytes of the string constant are shown in the listing, it is placed the machine code file in its entirety.

The following examples use the define byte (DFB) directive as described later in this manual. Very simply, the define byte (DFB) directive places the byte by byte value of the expressions on its right into the machine code file, starting at the present memory location shown on the far left of the listing.

```
0000                    ORG  0000H

                ;Valid examples of string constants are:
                ;
0000 5965612C20         DFB  "Yea, yea, yea!"
000E 41426162           DFB  "AB","ab"
0012 5468652064         DFB  "The dog Said 'Woof Woof!'"
```

4.3.3. Arithmetic Expressions

Cross-16 will accept arithmetic expressions made up of labels, integer constants, script brackets {} and arithmetic operators. Most of the arithmetic operators and their precedence are taken from the C programming language. A list of arithmetic operators, grouped in decreasing precedence, follows, where x and y represent integer values:

```
      $     Present value of program counter

    { }     script brackets

     ~ y    Ones complement of y

   INV y    Reversed (INVerted) byte order of y

     - y    Unary subtraction or twos complement of y

     + y    Unary addition of y

x    * y    multiplication of x and y

x    / y    division of x by y

x    % y    remainder after division of x by y
            (modulus)

x    + y    addition of x and y

x    - y    subtraction of y from x

x   << y    Left shift of x by y bits (y < 32)

x   >> y    Right shift of x by y bits (y < 32)

x    & y    Bitwise AND of x and y

x    ^ y    Bitwise XOR of x and y

x    | y    Bitwise OR of x and y
```

Some examples of expressions follow:

```
000100                  ORG     0100H
000100                  HOF     "INT16"

                        ;Arithmetic expressions grouped in decreasing precedence:
                        ;
00000100    =           EXP_:   EQU     $               ;Program Counter
                        ;
0000016C    =           EXP1:   EQU     4*("A"+26)      ;Script Brackets
                        ;
FFFFFFF0    =           EXP2:   EQU     ~15             ;One's complement
FFFFFFF1    =           EXP3:   EQU     -15             ;Two's complement
0000000F    =           EXP4:   EQU     +15             ;Unary addition
78563412    =           EXP5:   EQU     INV 12345678H   ;Invert
                        ;
00000FF0    =           EXP6:   EQU     0xff * 16       ;Multiplication
0000000F    =           EXP7:   EQU     0xff / 16       ;Division
                        ;
0000000F    =           EXP8:   EQU     0xff % 16       ;Remainder (Modulus)
                        ;
0000003C    =           EXP9:   EQU     40 + 20D;Addition
00000014    =           EXP0:   EQU     40 - 20D;Subtraction
                        ;
00123400    =           EXPa:   EQU     1234H << 8      ;Left Shift
00000012    =           EXPb:   EQU     1234H >> 8      ;Right Shift
                        ;
00000003    =           EXPc:   EQU     "3" & 15;Bitwise AND
                        ;
00000001    =           EXPd:   EQU     10B ^ 3         ;Bitwise XOR
                        ;
000000FF    =           EXPe:   EQU     2 | 253         ;Bitwise OR
                        ;
                        ; A little of everything
                        ;
0000001F    =           EXPf:   EQU     8 | 7 ^ 16 & 15 << 13 - 135/~(1-17)     ;1F

000000                  END
```

5. Cross-16 Assembler Directives

Cross-16 supports many common assembler directives combined with several of its own. These assembler directives should be considered as reserved keywords, and must not be duplicated by the processor nmemonics defined in the processors table. The directives are listed below:

```
CPU             CPU table declaration
DFB             DeFine Byte
DFS             DeFine data Storage
DFL             DeFine Long integer
DWM             Define Word (High byte first)
DWL             Define Word (Low byte first)
END             END of program
EQU             EQUate label to permanent value
HEX             Hexadecimal file on/off
HOF             Hexadecimal Output Format
INCL            INCLude source file
LIST            Listing on/off
ORG             ORiGin
PAGE            PAGE control
TITL            TITLe on listing
```

5.1. CPU - Central Processing Unit Table Declaration

The CPU directive tells Cross-16 which processor instruction table is to be loaded during assembly.

The CPU directive has the following syntax:

```
label:     CPU  "cpu_file_name"    ;comment
```

Only the instruction table file name may be specified after the CPU directive. A disk drive and/or directory may be included in the file specification. A label may be placed before the CPU directive, which will be assigned the current value of the program counter. The instruction table is only read once by Cross-16 during assembly, and all subsequent CPU directives will be ignored. An invalid CPU file name will result in a fatal error.

```
                ;Examples of the CPU directive
                ;
0043                    CPU     "1802.TBL"    ; CPU TABLE
0043                    CPU     "B:8048.TBL"  ; CPU TABLE
```

5.2. DFB - Define Byte

The define byte (DFB) directive allows the user to define the value of storage areas on a byte by byte basis.

The DFB directive has the following syntax:

```
label:    DFB  expr1,expr2,...,expr(n)  ;comment
```

Except for a string constant, the result of each expression must represent an 8 bit value (-128 to 255) or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. There is no limit on the number of bytes that may be defined using a single DFB directive, except that the length of an Cross-16 input line must not exceed 255 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

```
2000                    ORG  2000H

                ;   Valid examples of the DFB directive are:
                ;
2000 5061746965NOTE:    DFB  "Patient Lead Off",13,10
                                        ;ASCII string,CR,LF
2012 000102             DFB  0,1,2      ;Integers
2015 48                 DFB  77Q + 9    ;Expression
2017 3F44               DFB  3FH, 44H   ;Hexadecimal
```

5.3. DFL - Define Long Integer

The define long integer (DFL) directive allows the user to define the value of storage areas on a long integer or long word basis (a long integer is 32 bits or four bytes). There is no limit on the number of long integers that may be defined using a single DFL directive, except that the length of a Cross-16 input line must not exceed 255 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

The DFL directive has the following syntax;

```
label:     DFL  expr1,expr2,...,expr(n)   ;comment
```

Cross-16 must be able to represent the value of each expression using a 32 bit signed integer or an assembly error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 4 characters. The DFL directive stores 32 bit signed integers from the most significant byte to the least significant byte.

```
3000                    ORG  3000H

                ;Valid examples of the DFL directive are:
                ;
3000 00000002   CONST1:  DFL  2          ;Numeric constant
3004 00003000            DFL  CONST1     ;Label
3008 00000048            DFL  77Q + 9    ;Expression
300C 12345678            DFL  12345678H ;Hexadecimal
```

5.4. DFS - Define Storage

The define storage (DFS) directive may be used to reserve a section of memory with unspecified contents during assembly. This directive is often used to reserve an area of RAM or volatile memory for the target system. To insure that nothing is written to the hexadecimal file, this directive may be used with the HEX directive to turn the hexadecimal file off.

The DFS directive has the following syntax:

```
label:    DFS  expression     ;comment
```

The expression can be of any form which represents a 32 bit positive integer value, but only one expression is allowed. The value of the expression is added to the program counter and assembly continues. Except for the changed value of the program counter, no values are written to the hex file.

```
5000                    ORG  5000H

                ;Valid examples of the DFS directive are:
                ;
5000            STOR_B:  DFS  20 * 1    ;Reserve 20 bytes
5014            STOR_W:  DFS  20 * 2    ;Reserve 20 words
503C            STOR_L:  DFS  20 * 4    ;Reserve 20 long
```

5.5. DWL - Define Word (Least Significant Byte First)

The define word (DWL) directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). There is no limit on the number of words that may be defined using a single DWL directive, except that the length of a Cross-16 input line must not exceed 255 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

The DWL directive has the following syntax:

```
label:     DWL  expr1,expr2,...,expr(n)   ;comment
```

The result of each expression must represent a 16 bit integer value (-32768 to 65535) or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 2 characters in length. The DWL directive will store the least significant byte of the 16 bit value before the most significant byte. By using either the DWM or DWL directives, words may be placed in memory in a format which corresponds to the format used by the target processor.

```
4000                    ORG  4000H

                ;Valid examples of DWL directive are:
                ;
4000 0000010002CONST2:  DWL  0,1,2     ;Numeric constants
4006 4800               DWL  77Q+9      ;Expression
400A 3F004400           DWL  3FH,44H    ;Hexadecimal
400E FFFF               DWL  0FFFFH     ;up to 16 bits
```

5.6. DWM - Define Word (Most Significant Byte First)

The define word (DWM) directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). There is no limit on the number of words that may be defined using a single DWL directive, except that the length of a Cross- 16 input line must not exceed 255 characters. Although only the first 5 or 7 bytes of data will be shown in the listing, it is included in the hex file in its entirety.

The DWM directive has the following syntax:

```
label:   DWM  expr1,expr2,...,expr(n)  ;comment
```

The result of each expression must represent a 16 bit integer value (-32768 to 65535) or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 2 characters in length. The DWM directive will store the the most significant byte of the 16 bit value before the least significant byte. By using either the DWM or DWL directives, words may be placed in memory in a format that corresponds to the format used by the target processor.

```
3000                        ORG  3000H

                  ;Valid examples of the DWM directive are:
                  ;
3000 0000000100CONST1:   DWM  0,1,2     ;Numeric constants
3006 3000                DWM  CONST1    ;Label
3008 0048                DWM  77Q+9     ;Expression
300C 003F0044            DWM  3FH,44H   ;Hexadecimal
3010 FFFF                DWM  0FFFFH    ;up to 16 bits
```

5.7. END - End of Source Program

The end of assembly (END) directive is optional, but when used it will be the last line of the assembly source file assembled (the remainder being ignored).

This directive has the following syntax:

```
label:    END  expression      ;comment
```

The expression is optional, but will represent any positive 16 or 24 bit integer value, depending on the hexadecimal output format in use. When an expression is given, its value will be included as the address in the final line of the Intel and Motorola hexadecimal machine code file, representing the starting address of the assembly program. If the END directive or expression are not included, this starting address will default to zero.

```
                ;Valid examples of the END directive are:
                ;
0000                    END             ;Simple format
0000            THE_END: END  RESET     ;With starting address
0100                    END  0100H      ;Famous starting address
```

5.8. EQU - Equate Label

The equate (EQU) directive may be used to assign an integer value to a label.

The EQU directive has the following syntax:

```
label:    EQU  expression      ;comment
```

Neither the label or the expression are optional in this directive. The expression may consist of any numeric constant, character string or formula whose value can be represented in 32 bits. Cross-16 will place the value of expression in the label field followed by an equal sign "=" to show that this value is not the current value of the program counter. Defining a label more than once, or placing multiple expressions after the EQU directive, will result in an assembly error.

```
                ;Valid examples of the EQU directive are:
                ;
0013 =          CR:       EQU  13H       ;ASCII Carriage Return
000A =          LF:       EQU  10D       ;ASCII Line Feed
5012 =          CNTR:     EQU  $         ;Program counter
1300 =          EXPR2:    EQU  CR SHL 8  ;Formula
```

5.9. HEX - Hexadecimal File Control

The hexadecimal file control directive enables the user to turn the output to the machine code file "ON" or "OFF". A machine code file is not produced at all if a file name is not specified in the command line (-H TEST.HEX).

The HEX directive has the following syntax:

```
label:     HEX   "mode"          ;comment
```

where "mode" may be "ON" of "OFF". A label may be included with the HEX directive, which will be assigned the current value of the program counter. The HEX directive is optional, with the default mode at the beginning of each pass being "ON". The HEX directive may appear anywhere in the assembly source file. It is usually used when defining locations in RAM memory which the user does not want included in the machine code file.

```
                ; Valid use of HEX directive
                ;
0000                    ORG   0000H       ;Origin

0000                    HEX   "OFF"       ;TURN HEX FILE OFF

                BYTE1:  DFS   1
                BYTE2:  DFS   1
                BYTE3:  DFS   1
                WORD1:  DFS   2
                WORD2:  DFS   2
                WORD3:  DFS   2

0009                    HEX   "ON"        ;TURN HEX FILE ON
```

5.10. HOF - Hexadecimal Output Format

The hexadecimal output format (HOF) directive selects the format of the machine code output file.

The HOF directive has the following syntax:

```
label:     HOF  "format"     ;comment
```

A label may be included with the HOF directive, which will be assigned the current value of the program counter. The HOF directive is optional, and if not included Cross-16 will default to the "INT16" format. The HOF directive may appear anywhere in the assembly source file. If it appears more than once with different hexadecimal formats specified, the format of the hexadecimal file will change without an error code being generated.

The HOF directive partially controls the format of the assembled listing. If a HOF directive in not used, or one of the 16 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 32 bit value preceding the EQU and SETL directives, 24 bit addresses, and up to 7 bytes of code on each line. If one of the 8 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 16 bit value preceding the EQU and SETL directives, 16 bit addresses, and up to 5 bytes of code on each line. This allows the listing format to correspond to the address word of the target processor.

The machine code output file may be written in three different formats, binary, Intel and Motorola. Although only the Intel and Motorola formats are actually ASCII hexadecimal, the words hexadecimal and hex are used in this document to refer to all three formats.

```
  6500              ORG   6500H

                  ;Valid examples of the HOF directive are:
                  ;
006500                      HOF  "BIN16"  ;Binary 16 bit hex
6500              HOF  "BIN8"       ;Binary 8 bit hex
006500                      HOF  "INT16"  ;Intel 16 bit hex
6500              HOF  "INT8"       ;Intel 8 bit hex
006500                      HOF  "MOT16"  ;Motorola 16 bit hex
6500              HOF  "MOT8"       ;Motorola 8 bit hex
```

5.11. INCL - Include Source File

The include file (INCL) directive tells Cross-16 to insert a specified source file into the one specified in the command line, or a previous include file.

The INCL directive has the following syntax:

```
label:     INCL  "source_file_name"     ;comment
```

Included source files may only be nested a maximum of 4 deep before a fatal error occurs. A disk drive and/or directory may be included in the file specification. A label may be placed before the INCL directive, which will be assigned the current value of the program counter. All included files are read once each pass, and are included in there entirety in the listing. Should an END directive be encountered in an included file, assembly of all source files will cease at that point. A nonexistent include file will result in a fatal error.

```
                    ;An example of a INCL directive is:
                    ;
23A4                      INCL     "A:IO.ASM"
```

5.12. LIST - List File Control

The list file control directive enables the user to turn the output to the list file "ON" or "OFF". A list file is not produced at all if a list file name is not specified in the command line (-L TEST.LST).

The LIST directive has the following syntax:

```
label:     LIST   "mode"          ;comment
```

where "mode" may be "ON" or "OFF". A label may be included with the LIST directive, which will be assigned the current value of the program counter. The LIST directive is optional, with the default mode at the beginning of each pass being "ON". The LIST directive may appear anywhere in the assembly source file. It is usually used when debugging a specific section of source code, and the entire listing is not desired.

```
                ; Valid use of LIST directive
                ;
0000                    ORG   0000H

0000                    LIST  "OFF"         ;TURN LIST FILE OFF
000B                    LIST  "ON"          ;TURN LIST FILE ON
```

5.13. ORG - Program Counter Origin

The origin (ORG) directive allows the user to specify the value of the program counter during assembly.

The ORG directive has the following syntax:

```
label:    ORG  expression      ;comment
```

The ORG directive may be used as often as desired, but Cross-16 will not flag areas that may be defined more than once in a single source file. The expression is not optional in this directive and may consist of any numeric constant, character string or formula whose value can be represented in a 16 or 24 bit positive integer, depending on which hexadecimal format has been declared using the HOF directive. Cross-16 will place the new value of the program counter in the address field. A missing expression or multiple expressions after the ORG directive will result in an assembly error being flagged.

```
                ;Valid examples of the ORG directive are:
                ;
0000            RESET:    ORG  0          ;A common beginning
0100                      ORG  0100H      ;A famous start
```

5.14. PAGE - Page Control

The page (PAGE) directive may be used to both eject the current listing page and set the number of lines in each listing page.

In the following format:

```
label:    PAGE                    ;comment
```

Cross-16 will insert a form feed (0CH) into the listing before the next line of the listing. This causes the printer to continue the listing on a new page.

However, the next format:

```
label:    PAGE      expression        ;comment
```

will set the page length to the value of the expression. Each time the page length is reached, Cross-16 inserts a form feed into the listing. All positive integer values except 1 and 2 are legal. If the expression has a value of zero, or a page size is not specified, form feeds will not be inserted into the listing.

```
A000                    ORG    0A000H

                ;Valid examples of the PAGE directive are:
                ;
A000                    PAGE    56
A000                    PAGE     0  ;No form feeds
A000                    PAGE    60
A000                    PAGE       ;Form feed before next line
```

5.15. TITL - Title of Listing

The title (TITL) directive will cause Cross-16 to place the character string following the directive at the top of each page of the listing.

The TITL directive has the following syntax:

```
label:    TITL   "character string"     ;comment
```

Both the label and the comment are optional for this directive. The "character string" is not, however, and must at the very minimum be a null string "". There is no limit to the length of the character string, except that a Cross-16 input line cannot exceed 255 characters. Cross-16 does not require a TITL directive to produce a correctly formatted listing.

```
A800                    ORG  0A800H

                ;Valid examples of the TITL directive are:
                ;
A800                    TITL  "Cross-16 Test File"
A800                    TITL  "User's Choice"
```

5.16. WDLN - Word Length

The word length (WDLN) directive may be used to change the program counter word length from its default value of one, to any positive integer from one to eight inclusive. It is used by the TMS320 digital signal processor family which has a two byte, or sixteen bit, program word.

The WDLN directive has the following syntax:

```
label:     WDLN  expression      ;comment
```

The label is optional with the WDLN directive, and will be assigned the current value of the program counter. The expression may be any numeric constant with a value between one and eight inclusive. Missing or multiple expressions after the WDLN directive, will result in an assembly error being flagged.

Using the WDLN directive with the TMS320 family:

```
                    ;
0000                      CPU     "TMS320.TBL"     ; CPU TABLE
0000                      HOF     "INT8"           ; HEX OUTPUT FORMAT
0000                      WDLN     2               ; 2 BYTE WORD LENGTH
                    ;
                    ;     ORG     0
                    ;
0000 7E01           INIT: LACK    1
0001 7E21                 LACK    33
0002 7F8C                 CALA
0003 7F8D                 RET
                    ;
0000                      END
```

WARNING: When the WDLN assembler directive is used to set the program word length to a value other than 1 byte, Cross-16 may produce Intel and Motorola hexcode unlike what the user expects. In particular, the program counter and number of bytes in each line of the hex file have been changed to correspond on a one to one basis. Therefore, an eight bit EPROM can be programmed normally. Also a hex file splitter, (used to create a 16-bit EPROM from two 8-bit ones), should also work properly. These hexcode standards should be used with extreme caution, when the word length is not set to one (1). The binary format is not affected.

6. The Cross-16 Listing File

If requested using the -L directive in the command line, Cross-16 will produce a listing file during the second and third passes of the assembly source file.

The HOF directive partially controls the format of the assembled listing. If a HOF directive in not used, or one of the 16 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 32 bit value preceding the EQU directive, 24 bit addresses, and up to 7 bytes of code on each line. If one of the 8 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 16 bit value preceding the EQU directive, 16 bit addresses, and up to 5 bytes of code on each line. This allows the listing format to correspond to the address word of the target processor.

6.1. Listing Format

The listing is the original assembly source file, with 16 or 24 additional ASCII characters inserted at the beginning of each line. The next four or six characters represent the hexadecimal value of the program counter. Following the program counter, is another blank, followed by the hexadecimal value of the assembly instruction or assembler directive. This value will not be displayed after the fifth or seventh byte, but will be placed in the hexadecimal file in its entirety.

The EQU assembly directive is an exception to this rule, and further information on it may be found in chapter 5 of this manual.

6.1.1. Assembly Error Codes

When Cross-16 detects an error during assembly, a second, error message line will be placed immediately following the offending assembly line using the following format:

```
file(row,column): message
```

Where...

file is the name of the source file in which the assembly error was detected.

row is the line number of the source file in which the assembly error was detected.

column is Cross-16's character position on the source file line when the assembly error was detected.

message is Cross-16's explanation of the assembly error.

Many programming editors, such as the Microsoft Editor supplied with many of Microsoft's compilers, can use this format directly, giving the Cross-16 user an integrated development environment. The error code and assembled line are also shown on the video display, so that the user does not have to search the listing (or even generate it) to check for assembly errors. As Cross-16 searches the instruction table, it will display the first error code generated for the given assembly line, even though other errors may occur. Therefore, certain combinations of instructions and syntax errors can produce a misleading error code. If the code does not seem to be relevant, look for other possible errors. The assembly errors, their meanings, and an example of each, follow:

```
 1000                   ORG     1000H
 1000                   HOF     "INT8"
 1000                   CPU     "8048.TBL"

                ;Error  26:     Missing operand
 1000                   DFB                     ;Missing Operand
TEST.ASM(6,32): Error 26 - Missing operand

                ;Error  27:     Illegal line number
 1000           0G:                             ;Illegal line number
TEST.ASM(9,3): Error 27 - Illegal line number

                ;Error  28:     A "Character string" is required
 1000                   CPU     INT8            ;Character String
TEST.ASM(12,17): Error 28 - A "Character string" is required

                ;Error  29:     Missing or illegal label
 0001 =                 EQU     1               ;Missing label
TEST.ASM(15,1): Error 29 - Missing or illegal label

                ;Error  30:     Illegal hexadecimal format
 1000                   HOF     "TEK8"          ;Illegal hex format
TEST.ASM(18,17): Error 30 - Illegal hexadecimal format

                ;Error  31:     Unexpected characters at end of line
 0001 =         LAB:    EQU     1,2             ;Unexpected characters
TEST.ASM(21,20): Error 31 - Unexpected characters at end of line

                ;Error  32:     Phase error, value of label changes
 0001 =         LAB1:   EQU     1               ;Phase error
TEST.ASM(24,18): Error 32 - Phase error, value of label changes
 0002 =         LAB1:   EQU     2               ;Phase error
TEST.ASM(25,18): Error 32 - Phase error, value of label changes

                ;Error  33:     Instruction not found,
 1000 0000              MOV     A               ;Instruction not found
TEST.ASM(28,15): Error 33 - Instruction not found
```

```
                ;Error   34:     File control must be ON or OFF
 1002                   LIST    "YES"           ;File control must be ON or OFF
TEST.ASM(31,17): Error 34 - File control must be ON or OFF

                ;Error   35:     Symbol not found,
 1002                   XXX                     ;Instruction not found
TEST.ASM(34,33): Error 35 - Symbol not found

                ;Error   36:     Operand not in specified range
 1002 34                DFB     $1234           ;Too Large
TEST.ASM(37,21): Error 36 - Operand not in specified range



                ; The remaining errors only occur when Cross-16 is evaluating
                ; an expression


                ;Error   40:     Undefined label
 1003 00                DFB      + LABEL        ;Undefined label
TEST.ASM(46,19): Error 40 - Undefined label

                ;Error   41:     Missing " at end of character string
 1004                   LIST    "ON             ;File control must be ON or OFF
TEST.ASM(49,52): Error 41 - Missing " at end of character string

                ;Error   42:     Missing right script bracket }
 1004 00                DFB     4 * {7 - 3      ;Missing bracket
TEST.ASM(52,27): Error 42 - Missing right script bracket }

                ;Error   43:     Digit is not valid for declared base
 1005 3F                DFB     0779Q           ;Illegal nine
TEST.ASM(55,17): Error 43 - Digit is not valid for declared base

                ;Error   44:     Unexpected second value
 1006 00                DFB     1 2             ;Two values
TEST.ASM(58,18): Error 44 - Unexpected second value

                ;Error   45:     Undefined operator
 1007 00                DFB     1 ' 2
TEST.ASM(61,18): Error 45 - Undefined operator

                ;Error   46:     Unexpected right script bracket }
 1008 00                DFB     4 * {7 - 3}}    ;Extra bracket
TEST.ASM(64,28): Error 46 - Unexpected right script bracket }

                ;Error   47:     Unexpected end of line
 1009 00                DFB     4 *             ;End of line
TEST.ASM(67,20): Error 47 - Unexpected end of line
```

```
                ;Error   48:     Shift must be less than 32
 100A 00                DFB     2 << 33          ;Shift too large
TEST.ASM(70,24): Error 48 - Shift must be less than 32

                ;Error   49:     Unexpected binary operator
 100B 00                DFB     2 ** 33          ;Two operators
TEST.ASM(73,20): Error 49 - Unexpected binary operator

                ;Error   50:     Unexpected unary operator
 100C 00                DFB     33 ~             ;Wrong side
TEST.ASM(76,19): Error 50 - Unexpected unary operator

                ;Error   51:     String exceeds 4 character
 100D 42434445          DFL     "ABCDE"          ;Too long
TEST.ASM(79,17): Error 51 - String exceeds 4 character

                ;Error   52:     Unexpected expression separator
 1011 0001              DFB     4 * , 1          ;Unexpected comma
TEST.ASM(82,20): Error 52 - Unexpected expression separator

                ;Error   53:     Division by zero attempted
 1013 00                DFB     4 / 0            ;Division by zero
TEST.ASM(85,20): Error 53 - Division by zero attempted

 0000                   end
```

7. The Cross-16 Hexadecimal File

Cross-16 supports binary, Intel and Motorola 8 and 16 bit hexadecimal formats. The Intel 16 bit format "INT16" will be produced by default, and any one of the six may be selected using the HOF directive. The hexadecimal output file name is controlled using the -H directive in the Cross-16 command line. Examples of each are provided below for the assembly source file SHOW.ASM.

```
                ; File SHOW.ASM
                ;
A:SHOW.HEX              HOF  "INT8"          ;Hex Output Format
0100                    ORG  0100H
0100 4865786164BEGIN:   DFB  "Hexadecimal!" ;Character String
010C                    END  BEGIN
```

BIN8 -- Binary (8 bit format)

```
4865786164656369 6D616C21
```

BIN16 -- Binary (16 bit format)

```
4865786164656369 6D616C21
```

INT8 -- Intel (8 bit format)

```
:0C0100004865786164656369 6D616C217D
:00010001FE
```

INT16 -- Intel (16 bit format)

```
:020000020000FC
:0C0100004865786164656369 6D616C217D
:00010001FE
```

MOT8 -- Motorola (8 bit format)

```
S10F0100486578616465636 96D616C2179
S9030100FB
```

MOT16 -- Motorola (16 bit format)

```
S2100001004865786164656369 6D616C2178
S9030100FB
```

The machine code output file may be written in three different formats, binary, Intel and Motorola. Although only the Intel and Motorola formats are actually ASCII hexadecimal, the words hexadecimal and hex are used in this document to refer to all three formats. Notice that there is no difference in the hex code produced by the "BIN8" and "BIN16" hex types, only the format of the listing is changed. "BIN8" should be used unless the program counter exceeds 0FFFFH. Also note that the "BIN8" and "BIN16" hex output files are pure binary, not the ASCII representation shown above. Do not attempt to display one of these binary files on your screen or printer.

The binary hex code will start at the address of the first origin directive (ORG), unless it is preceded by an instruction, in which case the machine code starts at address location zero. Positive jumps in the program counter using the origin directive will be filled with the binary value 0FFH. If the program counter is reduced, the message

Warning -- Decreasing Program Counter In 'HEX' File

will appear on the screen and assembly will continue. Reducing the program counter using a binary hex code format can corrupt the hex file, and the user should either re-arrange the code or use another hex format.

WARNING: When the WDLN assembler directive is used to set the program word length to a value other than 1 byte, Cross-16 may produce Intel and Motorola hexcode unlike what the user expects. In particular, the program counter and number of bytes in each line of the hex file have been changed to correspond on a one to one basis. Therefore, an eight bit EPROM can be programmed normally. Also a hex file splitter, (used to create a 16-bit EPROM from two 8-bit ones), should also work properly. These hexcode standards should be used with extreme caution, when the word length is not set to one (1). The binary format is not affected.

8. The Cross-16 Instruction Table

The Cross-16 instruction table defines the nmemonics, operands and opcodes of individual microprocessors and microcontrollers. To simply use one of the tables provided, this section of the manual may be skipped, but the example assembly file for the target processor should be studied carefully. Cross-8, Cross-16 version 1.X, and Cross-32 earlier than version 1.5 users are advised that the Cross-16 version 2.0 table format is not compatible with these earlier products. Cross-32 tables may be converted to the Cross-16 V2.0 and later format, simply by changing the arithmetic operators to those used by the ANSI C programming language.

The instruction table consists of four sections, to define the processors registers, operands, addressing modes and mnemonics respectively. Each of these will be discussed as part of an example for the Intel MCS-48 (8048) microcontroller family. Although this is not the most world's most sophisticated processor, its instruction set has some unique attributes, which will exhibit most of the features and flexibility of the Cross-16 table format.

The Cross-16 processor tables are ASCII files, which may be edited using any text editor. Word processors should be avoided, or used in a non-document mode, to avoid the hidden formatting commands that are added to files by many of these products. The only limit on the size of the processor table, is that it must reside entirely in the RAM memory of your computer when Cross-16 is run. Any single line in the table must not exceed the maximum Cross-16 source line length of 255 characters. Cross-16 expects each line to end with an ASCII carriage return and line feed. Blank lines and comments may be entered at any point in the table using a semicolon ";", and are not stored in memory by Cross-16 during assembly.

8.1. Register Definition

The first section of the processor table defines the syntax of any registers or similar labels to which a constant value may be assigned. Cross-16 will use this section to identify operands as an exact character string. Each line of the register section has the following syntax:

```
register-number,"register0","register1",..."registerN"
      .
      .
      .
register-number,"register0","register1",..."registerN"
*
```

where:

Register-Number: is a unique integer, used by the operand section of the table to identify a register line. The line numbers need not be actual file line numbers, need not appear in numerical order, and may be assigned any positive integer value less than 32768. Tables supplied by Universal Cross-Assemblers will start with register line number one and increase in steps of one. Each line number must be followed by a comma ",".

Register: are character strings representing register names, defined like any other character string used with Cross-16, preceded and terminated by quotation marks ("), with each character string being separated by a comma ",". Cross-16 assigns the first character string the value of zero, and increases the value of each successive string by one.

***** Marks the end of the register section of the table.

The 8048 uses eight registers, which it labels R0 through R7 and assigns an address value of zero through seven respectively. The 8048 supports direct addressing of all eight registers, and indirect addressing registers R0 and R1. All eight directly addressed registers are defined by:

```
1,"R0","R1","R2","R3","R4","R5","R6","R7"
```

The two registers used for indirect addressing are defined by:

```
2,"R0","R1"
```

The end of the register section of the table is marked by:

```
*
```

8.2. Operand Definition

The second section of the table defines the size, format and position of operands in the instruction's opcode. The binary value of these operand definitions will be bitwise ORed with the instruction's opcode during assembly. Each line of the operand definition has the following syntax:

```
operand-number, start-bit, bit-length, expression, low, high
        .
        .
        .
operand-number, start-bit, bit-length, expression, low, high
*
```

where:

Operand-Number: is a unique integer, used by the addressing mode section of the table to identify this line. The operand numbers need not be actual file line numbers, need not appear in numerical order, and may be assigned any positive integer value less than 32,768. Tables supplied by Universal Cross-Assemblers usually start with line number one and increase in steps of one. Each line number must be followed by a comma ",".

Start-bit: is the position of the first bit of the operand within the instruction's opcode. This value must be a positive integer followed by a comma ",". The bit positions are numbered from left to right (most significant bit first) starting at 0 and preceding to a maximum of 255.

Bit-length: is the number of bits of the operand to be placed in the opcode, beginning with the least significant bit. The operand bit-length must be a positive integer ranging from 1 to a maximum of 32 bits.

Expression: is an integer arithmetic formula, whose value represents the operand to be incorporated into the opcode. This expression is not unlike any other used by Cross-16, with two exceptions:

1) A number sign "#" represents the original value of the operand found in the source file.

2) An at sign "@" immediately followed by a register-number points to a line in the register definition section of the table.

3) An opening single quote "`" represents the length of an instruction's hexcode in bytes. This is most often used with relative branch instructions. A tilde "~" was used for this purpose in versions of Cross-32 earlier than 1.5.

Low: is the smallest allowable value of the original operand found in the source file. If the operand value is less than the specified low, Cross-16 will look for another matching mnemonic located later in the table.

High: is the largest allowable value of the original operand found in the source file. If the operand value is greater than the specified high, Cross-16 will look for another matching mnemonic located later in the table.

***** Marks the end of the operand definition section of the table.

The 8048 instruction set uses five different operand types. The first two of these are part of the previously defined direct and indirect register addressing modes. In the direct register addressing mode, a three bit expression representing the register address is placed in the opcode starting at bit five. The character strings representing the registers were defined in register-number 1 of the register definition section. The direct register operand is defined by:

```
1,  5,  3,          @1,              0,                  7
```

Similarly, the indirect register operand, with its one bit value placed in bit seven of the opcode, is defined by:

```
2,  7,  1,          @2,              0,                  1
```

After that, the 8048's immediate operand, a signed eight bit value starting at bit 8 is defined by:

```
3,  8,  8,           #,           -128,                255
```

Next, the 8048 has an eight bit branch addressing mode. The operand in this mode is eight bits long and must be on the same 256 byte page as the next instruction {$+1}. This operand can be defined and checked using the expression:

```
4,  8,  8,    # & 255,    {$+1}& 3840,          {$+1}| 255
```

Finally, the 8048 has an eleven bit branch addressing mode. The operand in this mode is eleven bits long and be in one of the two memory banks. The most significant three bits of the operand are also shifted left five bits, to bits 0 through 2. However, even this operand can be defined and range checked using the following expression:

```
5,  0, 16,  {{# & 1792}*32}|{# & 255},   0,           4095
```

8.3. Addressing Mode Definition

The third section of the table defines the addressing modes of the processor and the position of the operands in the assembly language source code. In conjunction with the mnemonic definition in the next section, the addressing mode also defines the value and length of the opcode. Each line of the address mode definition section has the following syntax:

```
address-number, addressing-mode^hexcode:
      .
      .
      .
address-number, addressing-mode^hexcode:
*
```

where:

Address-Number: is a unique integer, used by the mnemonic section of the table to identify an addressing mode. The address-numbers need not be actual file line numbers, need not appear in numerical order, and may be assigned any unique positive integer less than 32,768. Tables supplied by Universal Cross-Assemblers start with address-number 1 and increase in increments of one. Each address-number must be followed by a comma ",".

Addressing-mode: is an ASCII string showing the relative positions of the characters and operands that make up the different addressing modes of a processor. The constant characters are simply listed. The variable operands are inserted using script brackets {} enclosing an operand-number as shown below:

```
{operand-number}
```

where the operand-number is defined in the operand section of the table. There are no limits on the character length of the addressing-mode-definition, except that no Cross-16 input line may exceed 255 characters.

Hexcode: is the ASCII hexadecimal representation of the instruction's opcode, that is placed between a carat "^" and a colon ":". From this and a similar field in the mnemonic section of the table, Cross-16 determines both the value and length of the instruction's opcode. There are no limits on the character length of the hexcode, except that a Cross-16 input line cannot exceed 255 characters. During assembly, the supplied hexcode is converted to a binary value and bitwise ORed with a similar field in the mnemonic section of the table and any operands defined

within script brackets "{}". The opcodes binary length is determined by the longer of the addressing and nmemonic hexcodes defined in the last two sections of the table.

* Marks the end of the addressing section of the table.

Most processor's instruction sets actually have more different instruction formats than manufacturer defined addressing modes. Each different instruction format that contains variable operands must be defined in this section of the table. When writing a table from scratch, examples of most of the processor's addressing formats can usually be found in the MOVE or LOAD instructions. The inherent addressing mode, which does not have any operands, such as the no operation (NOP) instruction used by many processors, is not defined in this section of the table. The 8048 has approximately eleven different addressing modes or formats including the inherent mode.

The first is the register addressing mode, where one of the processors eight registers is the operand, indicated by the {1}. The previous two sections of the table place the three bit operand at bit five of the one byte opcode and state that it must be R0 through R7.

```
1, {1}^08:
```

The second is the register indirect addressing mode, where either R0 or R1 is the operand, indicated by the {2}. The previous two sections of the table place the operand in bit seven of the one byte opcode and state that it must be R0 or R1.

```
2, @{2}^00:
```

The third is an immediate addressing mode, where an eight bit signed number is the operand, indicated by the {3}. The previous two sections of the table place the operand in the second byte of the two byte opcode and state that it must be in the range of -128 to 255.

```
3, #{3}^0000:
```

The remaining addressing formats are defined in a similar manner:

```
 4, {4}^0000:
 5, {5}^0000:
 6, @{2},A^00:
 7, {1},{4}^0000:
 8, {1},A^08:
 9, {1},#{3}^1800:
10, @{2},#{3}^1000:
*
```

8.4. Mnemonic Definition

The fourth and final section of the table defines the actual assembly mnemonics specified by the processor manufacturer, and the addressing modes used by each. In conjunction with the opcode definition in the previous section, the mnemonic section also defines the value and length of the instructions opcode. Each line of the mnemonic definition has the following syntax:

```
Mnemonic|address_number1-address_number2^hexcode:
      .
      .
      .
Mnemonic|address_number1-address_number2^hexcode:
*
```

where:

Mnemonic: is an ASCII string representing the instruction. Cross-16 uses a hash search routine to find the correct mnemonic for each line of the assembly language program. Therefore, the first (non-label) word of each assembly language source line must exactly match the first word defined in this section of the table, in both length and content. A word is a character string separated by white space (blank, tab or carriage return). After the first word, Cross-16 compares the source line with the mnemonic on a character by character, operand by operand basis. White space between the mnemonic and the first vertical line "|" is ignored. Each mnemonic should start with an alphabetic character "A-Z". Mnemonics do not have to be placed in any particular order. The mnemonics in tables supplied by Universal Cross-Assemblers are usually in alphabetical order for ease of reference. There are no limits on the character length of the mnemonic, except that a Cross-16 input line cannot exceed 255 characters.

Address_number: is the number of one of the addressing modes defined in the previous section of this table. Each address number is preceded by a vertical line "|" character. A range of consecutive address numbers may be defined by placing a hyphen "-" between two address numbers, i.e.

```
|1-4
```

Single address numbers and address number ranges may be mixed, as shown by the MOV line of the 8048 table:

```
MOV |6|8-10^A0:
```

Hexcode: is the ASCII hexadecimal opcode for the instruction that is placed between a carat "^" and a colon ":". From this field and a similar one in the previous section of the table, Cross-16 determines both the value and length of the instruction's opcode. There are no limits on the character length of the hexcode, except that a Cross-16 input line cannot exceed 255 characters. During assembly, the supplied hexcode is converted to a binary value and bitwise ORed with a similar field in the previous section of the table and any operands defined within the script brackets "{}". The total opcode length is determined by the longer of the addressing and mnemonic hexcodes defined in the last two sections of the table.

* Marks the end of the mnemonic section of the table.

There are three different addressing formats for the 8048's add to accumulator without carry instruction. The first word of this nmemonic is "ADD". This word starts the mnemonic line of the table, followed by a space. The second part of the nmemonic is an "A,", which is followed by a vertical line "|". The vertical line is followed by the addressing mode range. The "ADD A," nmemonic must be listed twice in the table, because the opcode for the immediate addressing mode bears no resemblance to that of the register and indirect modes.

```
ADD A,|1-2^60:
ADD A,|3^03:
```

When compiling an "ADD A" instruction, Cross-16 will search the table in the order that the addressing modes have been listed, i.e. 1, 2 and 3. If none of the addressing modes match, an assembly error will be flagged.

The remaining MCS-48 mnemonics are defined as follows:

```
ADDC A,|1-2^70:
ADDC A,|3^13:
ANL A,|1-2^50:
ANL A,|3^53:
ANL BUS,|3^98:
ANL P1,|3^99:
ANL P2,|3^9A:
ANLD P4,A^9C:
ANLD P5,A^9D:
ANLD P6,A^9E:
ANLD P7,A^9F:
CALL |5^1400:
CLR A^27:
CLR C^97:
```

```
CLR F0^85:
CLR F1^A5:
CPL A^37:
CPL C^A7:
CPL F0^95:
CPL F1^B5:
DA A^57:
DEC A^07:
DEC |1^C8:
DIS I^15:
DIS TCNTI^35:
DJNZ |7^E800:
EN I^05:
EN TCNTI^25:
ENT0 CLK^75:
IDL^01:
IN A,P1^09:
IN A,P2^0A:
INC A^17:
INC |1-2^10:
INS A,BUS^08:
JB0 |4^12:
JB1 |4^32:
JB2 |4^52:
JB3 |4^72:
JB4 |4^92:
JB5 |4^B2:
JB6 |4^D2:
JB7 |4^F2:
JC |4^F6:
JF0 |4^B6:
JF1 |4^76:
JMP |5^0400:
JMPP @A^B3:
JNC |4^E6:
JNI |4^86:
JNT0 |4^26:
JNT1 |4^46:
JNZ |4^96:
JT0 |4^36:
JT1 |4^56:
JTF |4^16:
JZ |4^C6:
MOV A,PSW^C7:
MOV A,T^42:
MOV A,|1-2^F0:
MOV A,|3^23:
MOV PSW,A^D7:
MOV T,A^62:
MOV |6|8-10^A0:
MOVD A,P4^0C:
```

```
MOVD A,P5^0D:
MOVD A,P6^0E:
MOVD A,P7^0F:
MOVD P4,A^3C:
MOVD P5,A^3D:
MOVD P6,A^3E:
MOVD P7,A^3F:
MOVP A,@A^A3:
MOVP3 A,@A^E3:
MOVX A,|2^80:
MOVX |6^90:
NOP^00:
ORL A,|1-2^40:
ORL A,|3^43:
ORL BUS,|3^88:
ORL P1,|3^89:
ORL P2,|3^8A:
ORLD P4,A^8C:
ORLD P5,A^8D:
ORLD P6,A^8E:
ORLD P7,A^8F:
OUTL BUS,A^02:
OUTL P1,A^39:
OUTL P2,A^3A:
RETR^93:
RET^83:
RL A^E7:
RLC A^F7:
RR A^77:
RRC A^67:
SEL MB0^E5:
SEL MB1^F5:
SEL RB0^C5:
SEL RB1^D5:
STOP TCNT^65:
STRT CNT^45:
STRT T^55:
SWAP A^47:
XCH A,|1-2^20:
XCHD A,|2^30:
XRL A,|1-2^D0:
XRL A,|3^D3:
```

8.5. Additional Table Features

There are numerous instruction tables included with Cross-16 which may be studied to gain further insight into the process of creating one. In addition to using the provided tables, Cross-16 provides many other options. The simplest, is that the provided tables may be modified to generate a cross-assembler that is more specific to a particular processor, rather than one that is for a family of processors. This is usually just a matter of deleting or adding several instructions. Therefore, as manufacturers produce new processors based on older processor families, Cross-16 will not become obsolete. Another feature is the ability to alter the manufacturers assembly mnemonics to a format which better suits the user's specific needs or preferences. A single set of common instruction mnemonics could be developed for example, and by only changing the opcodes in the instruction table, these same mnemonics could be assembled into machine language for a number of different processors. Only instructions that are extremely hardware specific would have to be re-examined. Using similar methods, Cross-16 can be used to ease software upgrading to a more recent processor, especially if both processors are from the same manufacturer, and have similar instruction sets.

8.6. Checking the Instruction Table

Cross-16 processor tables do not have to be sorted after being created or modified by the user. They should be double checked for accuracy. Any processor table format errors found during assembly will be displayed as a fatal error on the screen.

9. Non-Assembly Error Messages

During the assembly of a source file, Cross-16 may generate a "Fatal Error" or "Warning" message. Usually these messages will be self explanatory. Warning messages will not stop assembly, and are a result of a syntax error in the command line or a phase error. Encountering a fatal error will cause Cross-16 to terminate execution. When this occurs, all open files are closed, a fatal error message is sent to the screen and Cross-16 exits to the operating system.

The following example illustrates a warning message when an illegal option "-O" is placed in the Cross-16 command line. The error count "No Errors" refers to assembly errors only.

```
C>C16 E8085.ASM -O E8085.HEX


Cross-16 Meta-Assembler   PC/MS-DOS   Version 2:01
Copyright (C) 1989 Universal Cross-Assemblers

Warning -- Illegal Option Ignored

Warning -- Extra Source File Ignored

Starting pass number 1

Starting pass number 2

Checksum = 4212
End of Assembly -- No Errors

C>
```

The following example illustrates a fatal error when a source file is placed in the Cross-16 command line which does not exist.

```
C>C16 NOFILE.ASM


Cross-16 Meta-Assembler   PC/MS-DOS   Version 2:01
Copyright (C) 1989 Universal Cross-Assemblers

Source file "NOFILE.ASM"
Fatal Error -- Source file did not open


C>
```

9.1. Warning Messages

Warning messages will not stop assembly, and are a result of a syntax error in the command line or a phase error. When such a problem occurs, "Warning -- " will appear on the screen, followed by the appropriate message. The message will be one of the following:

Illegal Option Ignored

Cross-16 only supports the -L and -H command line options. All others will be ignored, as will any file name that follows them.

-H Option Ignored -- Missing File Name

A hexadecimal output file has been requested, but a file name has not been provided. Assembly continues, but a hex file is not produced.

-L Option Ignored -- Missing File Name

A listing has been requested, but a file name has not been provided. Assembly continues, but a listing is not produced.

Decreasing Program Counter In 'HEX' File

An origin (ORG) statement has reduced the value of the program counter in a source program in which either the "BIN8" or "BIN16" hex output format (HOF) has been specified. This can cause a corruption of the hex output file that the user had not intended.

Extra Source File Ignored

Only one source file can be specified in the command line. Additional files can be included via the INCL directive within the source file. This warning often follows the "Illegal Option Ignored" message.

Extra Hex File Ignored

Only one hexadecimal output file can be specified after the -H option in the command line. This warning often follows the "Illegal Option Ignored" message.

Extra List File Ignored

Only one listing can be specified after the -L option in the command line. This warning often follows the "Illegal Option Ignored" message.

Phase Error

One or more labels have been assigned a value in the second pass that was not equal to the assigned value in the first pass. Cross-16 will perform a third pass to correct this problem.

Some processors have instructions with identical syntaxes but different opcode lengths. Cross-16 assigns undefined labels a value of zero. This can cause Cross-16 to use the instruction with the shortest opcode in the first pass, and an instruction with a longer opcode in the second pass. The 6805 for example has two jump instructions, the first only supports jumps to page zero (00H to 0FFH) while the second supports jumps to the entire address space (00H to 0FFFFH). A jump to an undefined address during the first pass will use the two byte opcode, while the same jump to a now defined address, which is not on page zero, will use the three byte opcode during the second pass. Assembly in this manner will produce the shortest, fastest machine code possible, but does require three passes. Users which are not pressed for either space or speed may wish to eliminate this third pass to reduce Cross-16's assembly time. This may be accomplished by defining memory labels at the beginning of the program, not using forward referenced labels and/or removing the the shorter opcodes from the instruction table. The final version of your software can always be reassembled with the shorter opcodes installed.

9.2. Fatal Errors

Encountering a fatal error will cause Cross-16 to terminate execution. When this occurs, all open files are closed, "Fatal Error -- " followed by an appropriate message appears on the screen and Cross-16 exits to the operating system. The message will be one of the following:

No Source File Specified

A source file name was not included in the command line.

Source File Did Not Open

The source file included in the command line did not open. The name of the offending file is displayed on your screen. The file name is incorrect or on a disk drive or directory other than the one specified.

List File Did Not Open

The specified list file name is on a nonexistent disk drive or directory, or the directory is full.

Hex File Did Not Open

The specified hex file name is on a nonexistent disk drive or directory, or the directory is full.

Disk or Directory Full

Delete unnecessary files and try again.

Too Many Include Files

Include files using the INCL directive may only be nested 4 deep.

Too Many Conditional Blocks

Conditional blocks using the IF, ELSE and ENDI directives may only be nested 16 deep.

Insufficient memory while loading table

Cross-16 has run out of memory while loading the instruction table. This will only occur using a table for a 16-bit processor on a PC/MS-DOS computer with less than 192kB of RAM, if the CPU directive is located a the beginning of the source file.

Insufficient memory while loading symbol

Cross-16 has run out of memory when attempting to store a label in RAM. Reduce the number or the length of the labels in the source file or add more memory. A PC/MS-DOS computer with 640kB should store a minimum of 19,000 labels.

Illegal CPU table format

Cross-16 has discovered a format error in the CPU table while loading it into RAM. The offending line is displayed on your screen. Correct the problem and run Cross-16 again.

10. Bugs

Should a possible bug or technical error be found in this program, please call, write or fax Universal Cross-Assemblers at the address on the first page of this manual.

Please specify the following:

Program and version:

Operating system and version:

Target processor:

Problem:

Sending a disk, complete with an example showing the problem, will result in the most complete resolution of your problem. All disks sent to Universal Cross-Assemblers will be returned, with an explanation of the problem and the latest revision of your version of the product.

11. Processor Manufacturers

The following is an alphabetical list of the manufacturers from whom further information may be obtained regarding the microprocessors and microcontrollers for which instruction tables are supplied by Universal Cross-Assemblers:

Hitachi America, Ltd. Semiconductor and IC Sales 2210 O'Toole Avenue San Jose, CA 95131	6303 64180
Intel Corp 3065 Bowers Avenue Santa Clara, CA 95051	8048 8051 8085 8096
Motorola Integrated Circuits 3501 Ed Bluestein Blvd Austin, TX 78721	6801 6805 6809 68HC11
National Semiconductor Corp 2900 Semiconductor Drive Santa Clara, CA 95051	COP400 COP800
RCA Solid State Division Rte 202 Somerville, NJ 08876	1802
Rockwell International Microelectronic Devices Div 4311 Jamboree Road Newport Beach, CA 92660	65C02
Texas Instruments Incorporated P.O. Box 1443 Houston, TX 77001	TMS320 TMS370
Zilog Inc 1315 Dell Avenue Cambell, CA 95008	Z8 Z80, Z180 SUPER8

12. Registration

All users who purchased this product directly from Universal Cross-Assembler are automatically registered. Others, who purchased the product through a distributor, or the purchasing department of a large institution, should return a copy of this page in the envelope provided to Universal Cross-Assemblers.

This will insure that you personally receive updates, discounts on future products, or other information published by Universal Cross-Assemblers.

Name: _____

Address: _____

_____

_____

_____

Telephone: _____

Facsimile: _____

Index

#
#, 41

$
$, 14

%
%, 14

&
&, 14

*
*, 14

+
+, 14

-
-, 14

/
/, 14

8
8048, 39

:
:, 43, 46

<
< <, 14

=
=, 24

>
> >, 14

@
@, 41

A
Abort
    assembler, **8**
Acknowledgement, **2**
Addition, 14
Address
    mode, **43**
    number, **43**, 45
    starting, 23
Addressing mode
    definition, **43**
AND, 14
Arithmetic expressions, **14**
ASCII, 6
Assembler, **5**
Assembly line, 9
At sign @, 41

B
Base
    number, **12**
Bit-length, **41**
Bugs
    reporting, **55**

C
C
    language, 11, 14
Case, **6**, 13
Checksum, **8**
Column, **33**
Command line, **7**
Comment, **9**
CPU, 17
CPU table
    illegal format, **54**
Cross-16, 39
Cross-32, 39
Cross-8, 39
Cross-assembler, **5**

D
Define
    long integer, **19**
    storage, **20**
Define byte, **18**
Define word, 21, 22
DFB, **18**
Directives
    assembler, **16**
Directory, 7
Disclaimer, **2**
Division, 14
DWL, **21**

DWM, **22**

E
END, **23**
table, 40
EPROM, 32, 38
EQU, **24**
Equate label, 24
Error
assembly, 33
phase, 52
Errors
fatal, **53**
non-assembly, **50**
warning, **51**
Exclusive OR, 14
Expression, 41
Expressions, **14**

F
Fatal
errors, **53**
File, 33
binary, 38
hexadecimal, 37
list, 28, 33
Form feed, 30

H
HEX, **25**
Hex file, 7
Hexadecimal, 5
file, 25
format, 26
Hexadecimal file, **37**
Hexcode, 43, **46**
High, **42**
HOF, **26**

I
INCL, **27**
Include file, 27
Inherent, 44
Installation, **7**
Instruction table, 17
Integer
16-bit, 21, 22
long, 19
Integer constant, **11**
INV, 14

J
Jumps
positive, 38

L
Label, 9
Labels, **10**
License, **2**
Line #, 9
Linker, 6
LIST, **28**
List file, 7
Listing, 8, **33**
format, 33
Low, 42

M
Manufacturers
processor, 56
MCS-48, 39
Memory
insufficient, **54**
Message, **33**
Meta-assembler, **5**
Mnemonic, 45
definition, 45
Mnemonics, **5**
Modulus, 14
Multiplication, 14

N
Number sign #, 41
Numeric constant, **11**

O
Operand
definition, 41
expression, 41
number, 41, **43**
Operands, 9, **11**
Operation, 9
Operators
arithmetic, 14
logical, 14
OR, 14
ORG, **29**
Origin, 29

P
PAGE, **30**
control, 30
length, 30
Phase error, **52**
Program counter, 20, **29**

Q
Quote
CPU table, **42**

R
Register
definition, **40**
numbers, 40
strings, 40
Registration, **57**
Remainder, 14
Reserved keywords, 16
Row, **33**
Run, **7**
Running Cross-32, **8**

S
Script brackets, 14, 44, 46
Shift
left, 14
right, 14
Source file, **9**
Start-bit, **41**
String constants, **13**
Subtraction, 14

T
Table
checking, 49
processor, 17
sorting, 49
Target processor, 5
TITL, **31**
Title
listing, 31

U
Unary, 14

W
Warning
messages, 51
WDLN, 32, 38
Warranty, **2**
WDLN, **32**
Word length, 32
Word processors, 39

X
X
C integer, 11
XOR, 14

^
^, 43, 46
XOR, 14

‘
‘
CPU table, **42**

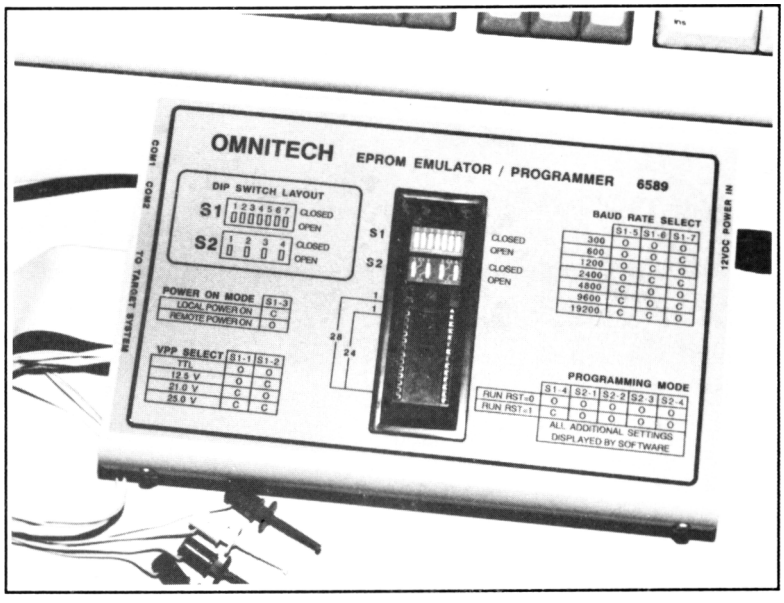
{
}, 14, 44, 46

|
OR, 14

~
~, 14

OMNITECH EPROM EMULATOR/PROGRAMMER 6589

The OMNITECH 6589 EPROM Emulator/Programmer eliminates the need for EPROM programming, debugging and erasing during the development of any micro-processor based system. The EPROM Emulator/Programmer is connected to the EPROM socket of the target system with an additional lead connected to the target system reset line. The assembled target system software that is normally down-loaded to an EPROM programmer is sent via any computer serial port to the OMNITECH 6589 EPROM Emulator/Programmer. The reset line of the target system is then released by user command allowing the target system CPU to execute the down-loaded program code.



A versatile command set for the OMNITECH 6589 EPROM Emulator/Programmer allows the developer to edit, search, copy or move memory contents using a terminal or a PC with communications software. Memory contents can be up-loaded or down-loaded in Motorola "S", Intel HEX and straight binary formats. The OMNITECH 6589 EPROM Emulator/Programmer contains battery backed up SRAM. This preserves the users code during unexpected power failures or when power is switched off.

A byte exchange feature allows the EPROM Emulator/Programmer to appear as a UART to the target system and allows the user to exchange ASCII characters or hexadecimal bytes with the target system in real time. This eliminates the need for LEDs or DIP switches as part of the target system input/output hardware. Further, it can be used to provide a target system register dump for debugging purposes or used as a single byte of dual port ram.

The OMNITECH EPROM 6589 Emulator/Programmer can program or copy the popular JEDEC 24 pin or 28 pin EPROMS or EEPROMS using the on-board programming socket. The contents of existing EPROMS or EEPROMS can be copied to internal memory, modified and reprogrammed or up-loaded to a PC. The EPROM contents are checked before, during and after programming to ensure data integrity.

The OMNITECH 6589 EPROM Emulator/Programmer includes a RUN mode. This mode automatically releases the target system reset line following a HEX file down-load and eliminates the need for operator command sequences.

An on-line HELP command displays summary information for all supported commands. For fast command entry, most commands have a single letter equivalent.

OMNITECH EPROM EMULATOR/PROGRAMMER 6589

Physical Specifications:

* Dimensions - 219mm L x 148mm W x 33mm H (8-5/8" x 5-3/4" x 1-1/4")
* Weight - 1 kg (2.2 lbs)

Electrical Specifications:

* Emulates, programs and verifies the following EPROMs (Generic P/Ns):
  2716, 27C16, 2716B, 2732, 27C32, 2732A, 2732B, 2764, 27C64, 2764A,
  27128, 27C128, 27128A, 27256, 27C256, 27512, 27C512.
* Standard 50ms, or Adaptive 1ms programming algorithm can be selected.
* 3 1/2 minutes average program time for 27512 using Adaptive algorithm.
* Loads the following EEPROMs (Generic P/Ns): 2816, 2864, 28256.
* Access time for simulated EPROM - 120 ns (user replaceable SRAM and interface).
* Supplied with CSA, UL approved 120V/60HZ AC adaptor (220V/50HZ adaptor available).
* Internal 512K SRAM (64Kx8) backed up by lithium battery.
* RS-232C serial interface via standard DB-25S female connector.
* Communications protocol - 8 data bits, 1 stop bit, no parity, half duplex.
* Baud rates supported 300, 600, 1200, 2400, 4800, 9600 and 19200.
* XON/XOFF and CTS/RTS handshaking is supported.
* Supplied with 18" ribbon cable, 24 and 28 pin DIP plugs for connection to target system EPROM socket.
* Byte exchange feature utilizes READ, WRITE, and ENABLE lines connected to the target system and appears as 2 RAM locations sharing the EPROM data buss.
* Target system reset line can be active HIGH or LOW.
* Complete users manual included.
* Warranty - 90 days parts and labor.

System Commands:

* EDIT, COPY, FILL, MOVE, DISPLAY, and SEARCH memory contents.
* SETUP system and EPROM parameters.
* RUN and STOP execution of target system.
* INPUT and OUTPUT Binary, Motorola "S" and INTEL Hex records.
* HELP displays commands with explanation of their function.
* PROGRAM and VERIFY EPROMS.

The OMNITECH 6589 EPROM Emulator/Programmer will operate from any "dumb" terminal or personal computer with terminal software.

Manufactured by:

Omnitech Incorporated
P.O.Box 2622, Station M
Halifax, Nova Scotia
Canada
B3J 3P7

Distributed by:

UNIVERSAL CROSS-ASSEMBLERS
P.O. Box 6158
SAINT JOHN, N.B. CANADA
E2L 4R6
Voice/Fax: (506) 847-0681